

## Inbetriebnahme

### Hardware

Die minimale Beschaltung zeigt Bild 1. Beim Einschalten blinkt das LED kurz auf und leuchtet dann dauernd. Der Jumper an /IRQ ist normalerweise nicht gesteckt. Reset kann mit dem Taster ausgelöst werden.

Es ist sicherer die 5V mit einem Spannungsregler ( 78L05 ) auf der Leiterplatte zu erzeugen als ein Labornetzgerät zu verwenden. Quarz und PLL-Schleifenfilter direkt neben dem IC anordnen und keine anderen Signale über ihren Masseanschluß führen. Geeignete Kondensatoren an Quarz und Filter verwenden: COG/ NP0 bzw. X7R oder Folie. Z5U-Kondensatoren vermeiden.

Für die V24 ist eine Sparversion gezeigt. Wenn man eine 9 Pin Buchse mit Modem-Pinbelegung auf die Leiterplatte setzt, kann man handelsübliches 9pol Kabel verwenden das alle Signale 1 : 1 durchverbindet. Sowie optional handelsüblichen 9 auf 25 Pin V24 Umsetzer, falls der PC 25 Pin Stecker hat.

### Terminalprogramm

Es sollte so ziemlich jedes gängige Programm verwendbar sein, eine Übersicht ist im Anhang ( Seite 3.x ) aufgeführt. Einstellung:

- \* 9600 Baud, 8N1
- \* Vollduplex, kein automatisches Echo vom Terminalprogramm

- \* Handshake XON/XOFF, Buffer auf minimale Größe einstellen
- \* Emulation TTY

Für Übertragung von Sourcefiles sollte das Programm Upload von ASCII-Files von Diskette und Download auf Diskette können. Es ist kein XMODEM, Kermit usw. nötig.

Wenn das Terminalprogramm läuft, wird nach Druck auf die

Resettaste am Controller diese Meldung erscheinen:

```
RESET
- - - |
```

Eingetippte Zeichen erscheinen dann als Echo am Bildschirm. Man beachte, daß Kleinbuchstaben automatisch auf Großbuchstaben gewandelt werden.

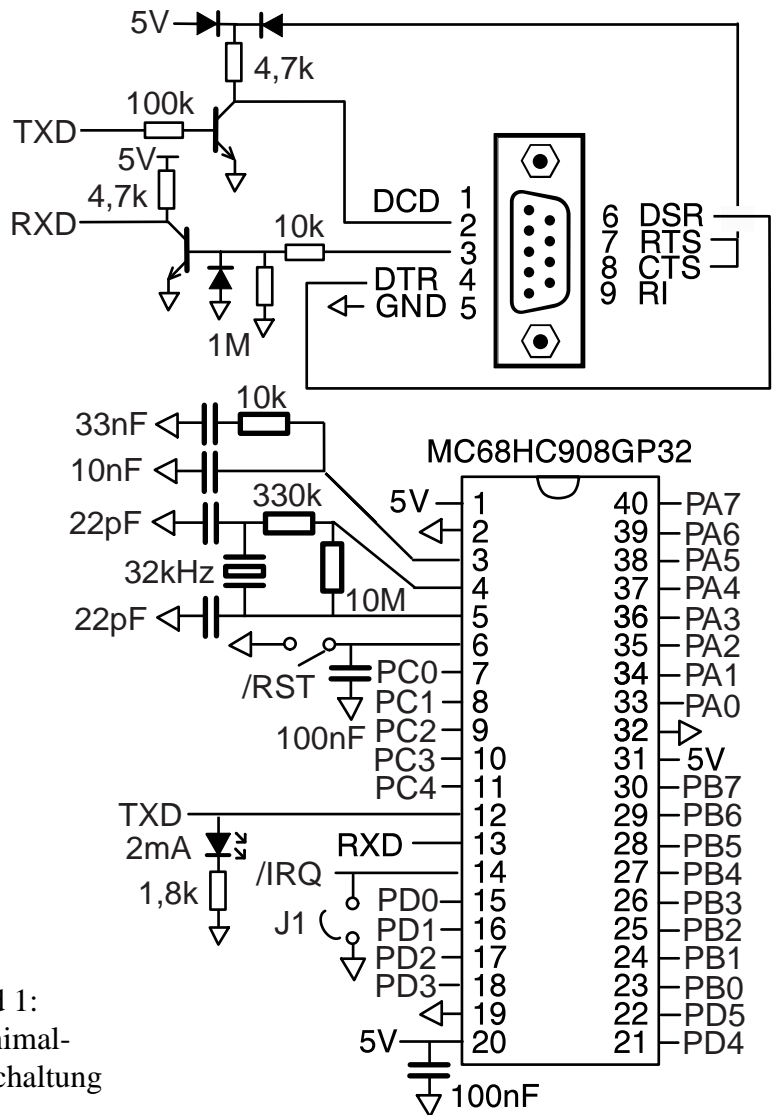


Bild 1:  
Minimal-  
beschaltung

Die Zahlenbasis im Handbuch genau wie im System ist normalerweise Hex, aber manchmal ist es angebracht eine Zahl dezimal darzustellen. Angehängtes h oder d sorgt in Zweifelsfällen für Klarheit:

1234h Hexzahl  
1234d Dezimalzahl

Der Buchstabe h als Platzhalter gibt auch die Länge eines Datenworts an:

hh 8 Bit  
hhhh 16 Bit

## FORTH

Der einfachste Zugang zu FORTH ist, wenn man sich vorstellt, daß ein virtueller 16 Bit Stackprozessor durch eine Sammlung von Assemblerunterprogrammen auf einer realen 8 Bit CPU simuliert wird. Diese Nucleus-Befehle findet man auf Seite 2.10 mit Speicherverbrauch und Ausführungszeit wie bei Opcodes einer CPU üblich aufgeführt. Man stellt auch fest, daß sie kaum mehr tun als das was man von Assembleropcodes erwartet: Daten bewegen, Arithmetik, boolesche Logik, Sprungbefehle usw..

Neu ist also nur, daß als virtueller Prozessor eine 0-Adreßmaschine, ein Stackprozessor gewählt wurde. Die gewohnte Infix-Notation wird durch Postfix (= RPN „reverse polish notation“) ersetzt. Praktisch genügt es aber den Befehl hinter den Operanden stellen:

Infix Postfix  
2 + 3 = 2 3 +

Das „=“ entfällt, weil + sofort ausgeführt werden kann, da alle Daten bereits vorliegen. Sie wurden automatisch auf dem Stack zwischengespeichert. Auch Klammern sind nicht mehr nötig, weil es genügt die Abfolge geeignet anzuordnen:

Infix Postfix  
2\*(3+4) = 3 4 + 2 \*

Aufgrund dieser Vorzüge arbeiteten die ersten Taschenrechner

von Hewlett Packard so. Auch der erste leistungsfähige Arithmetik Co-prozessor Intel 8087 war eine Stackmaschine. Selbst bei vielen Programmiersprachen die gegenüber dem Programmierer Infix verwenden, wandelt der Compiler in Postfix um.

Während die geänderte Notation des Stackprozessors anfangs trotzdem eher als lästig empfunden wird sind daraus folgende Eigenschaften um so vorteilhafter.

Eine 0-Adreßmaschine hat keine Adressierungsarten. Die Opcodelist des Stackprozessors auf Seite 2.10 ist deshalb deutlich kürzer als die des 68HC08 ab Seite 3.9. Und damit besonders leicht erlernbar.

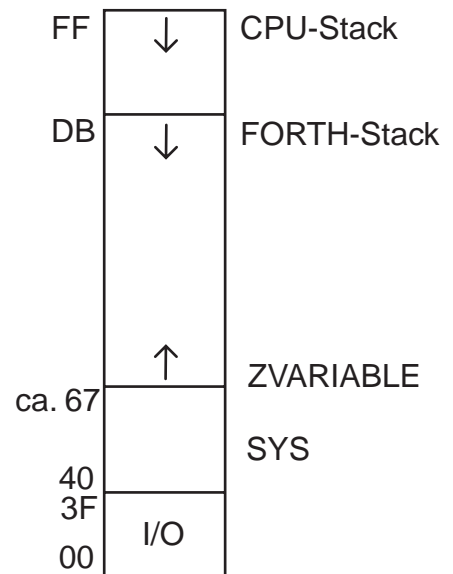
Die Datenschnittstelle für den virtuellen Prozessor ist einheitlich und simpel der Stack. Über den kann der Programmierer Daten übergeben und Ergebnisse entnehmen. Es ist deshalb besonders einfach diesen Prozessortyp als Interpreter von einem Terminal aus interaktiv zu betreiben.

Da der Stack auch fast alle lokalen Variablen ersetzt, sind Unterprogramme isoliert testbar. FORTH-Programme werden deshalb aus vielen kleinen, leicht prüfbar Unterprogrammen zusammengesetzt. Das ist möglich, weil die Opcodes des virtuellen Prozessors zwar besonders auf dem 68HC08 relativ langsam sind, aber der Aufruf eines Unterprogramms in Geschwindigkeit und Speicherverbrauch dem Aufruf eines Assemblerunterprogramms entspricht. Es müssen keinerlei Stackframes aufgebaut werden.

Mit FORTH erzeugt man damit zwar nicht die schnellsten Programme. Aber unter Berücksichtigung der günstigeren Testphase schneller Programme.

Für die Programmteile in denen die Geschwindigkeit von FORTH nicht genügt, hat nanoFORTH einen Assembler und Disassembler für den 68HC08 eingebaut. Der Stack sorgt auch hier für eine klare und einfache Schnittstelle zwischen Hochsprache und Assembler. Teilprogramme in Assembler verhalten sich wie FORTH-Befehle und passen damit homogen ins Gesamtsystem.

Bild 2: Speicher  
ZeroPage RAM



## Speicher

Die wichtigsten Port- und Speicheradressen sind als Konstanten vordefiniert und damit im Programm unmittelbar verwendbar (Tabelle 1).

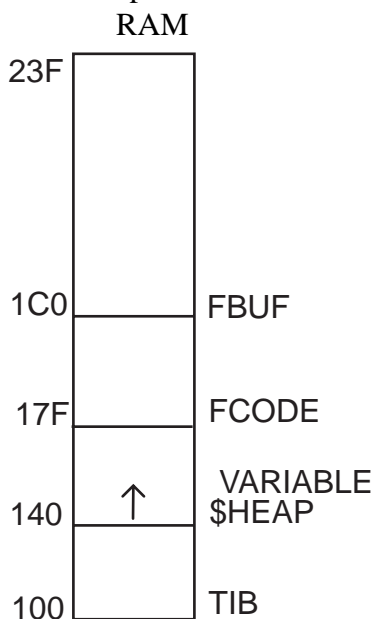
### RAM

Die CPU hat spezielle Adressierungsarten die den Zugriff auf die ZeroPage also den Bereich 0000 - 00FF besonders effizient machen (Bild 2). Deshalb liegen dort auch von 0000 - 003F die I/O Register. Im RAM liegen ab 0040 die Systemvariablen. Z.B. auch der N-Bereich mit 8 Bytes der als Arbeitsspeicher in Assemblerprogrammen verwendet wird. Es

Tabelle 1: Speicherkonstanten

00	CONSTANT PA	\ Port A
01	CONSTANT PB	
02	CONSTANT PC	
03	CONSTANT PD	
04	CONSTANT DPA	\ Dir PA
05	CONSTANT DPB	
06	CONSTANT DPC	
07	CONSTANT DPD	
0100	CONSTANT TIB	
0140	CONSTANT \$HEAP	
017F	CONSTANT FCODE	
01C0	CONSTANT FBUF	
8000	CONSTANT APP-EE	
8080	CONSTANT SYS-EE	
8100	CONSTANT APP	
B800	CONSTANT SYS	

Bild 3: Speicher



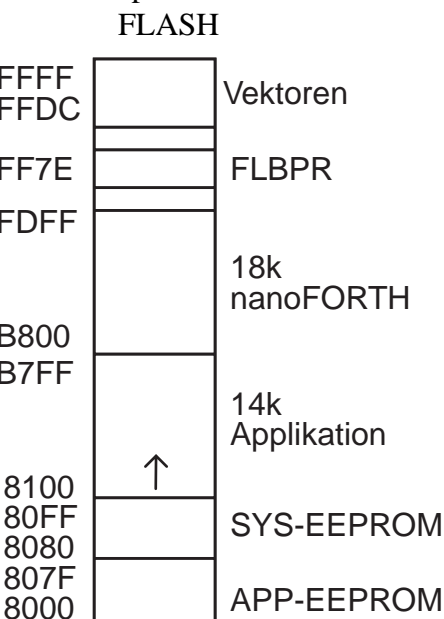
folgen für die Applikation mit ZVARIABLE neu definierte Variablen die aufwärts wachsen.

Den Variablen entgegen wächst abwärts der FORTH-Stack. Typisch genügen ihm ca. 16d Byte. Oberhalb ist für den CPU-Stack ein fester Bereich von 36d Bytes vorgesehen. Da für diesen auch meist 16d Bytes genügen kann man oberhalb des FORTH-Stacks gegebenenfalls noch mit CONSTANT etwas RAM für die Applikation belegen.

Wie tief die beiden Stacks tatsächlich typisch wachsen kann man leicht durch Ausdruck des Speichers mit z.B. 00C0 DUMP feststellen, da das komplette RAM nach Reset auf 00 initialisiert wird.

Enger gehts im RAM des Hauptspeichers ab 0100 zu (Bild 3). TIB („Terminal Input Buffer“) mit 64d Byte speichert die ASCII-Zeichen die von der V24 kommen. \$HEAP enthält während des Compilierens die Labels des Assemblers. FCODE wird zum Schreiben des FLASHs benötigt. Also für Schreiben des EEPROMs und Compilierung. FBUF wird nur für Schreiben des EEPROMs benötigt. Nach dem Compilieren bzw. Schreiben des EEPROMs stehen diese drei Bereiche als RAM für die Applikation zur Verfügung. Deshalb ist VARIABLE auf 140 initialisiert, damit dort aufwärts wachsend Variablen definiert werden können.

Bild 4: Speicher



## FLASH

Von den ca. 32k Byte FLASH sind 18k vom System belegt, während 14k für die Applikation bleiben (Bild 4). Die Applikation wächst von 8100 aufwärts,

FLASH ist für 10k Schreib & Löschzyklen spezifiziert, kann also praktisch beliebig oft umprogrammiert werden. In zwei 128 Byte Seiten wird per Software EEPROM simuliert. Eines für die Applikation. Das andere enthält System-Konstanten die man eventuell verändern will.

## EEPROM

Auf FF gelöscht FLASH ist zwar in einzelnen Bytes programmierbar. Aber nur in Seiten von 128 Byte löscher die an Seitengrenzen ausgerichtet sein müssen. Das Treiberprogramm für beide Funktionen muß im RAM stehen. Denn während des Zugriffs ist das FLASH nicht ansprechbar. Schreiben eines Bytes dauert ohne Overhead nur 40usec, aber Löschen 40msec.

Um ein Byte im EEPROM zu überschreiben muß der EEPROM-Bereich also erst ins RAM nach FBUF kopiert werden und dort muß das Byte überschrieben werden. Dann wird ein Programm nach FCODE kopiert und ausgeführt das die Seite im FLASH

löscht. Danach wird ein Programm nach FCODE kopiert das die Seite von FBUF zurück ins FLASH schreibt.

Um die Details muß sich der Anwender allerdings nicht kümmern, denn es gibt Befehle zum Schreiben der EEPROM-Bereiche:

```
EC! \ ( C1 Addr1 - )
      \ write Byte
E! \ ( N1 Addr1 - )
     \ write 16 Bit
```

Man ist mit den beiden Befehlen nicht auf Zugriff ins EEPROM beschränkt. Auch Bytes im Applikations-FLASH und bei gestecktem Jumper kann auch nanoFORTH verändert werden. Nicht jedoch FLBPR und die Vektoren.

Das simulierte EEPROM ist brauchbar für Konstanten die man oft lesen, aber nur im Interpretermodus manchmal ändern will. Wenn eine Applikation jedoch im Betrieb EEPROM ändern muß, ist die Verwendung eines externe seriellen EEPROMs günstiger. Man gewinnt dann FBUF und FCODE als RAM für die Applikation. Und hat auch keine Probleme mit Interrupts.

## Compiler

Um auf akzeptable Compilergeschwindigkeiten zu kommen ist der Programmspeicher normalerweise gelöscht. Dann kann man mit diesen Befehlen problemlos bytewise schreiben:

```
FC! \ ( C1 Addr1 - )
      \ write Byte
F! \ ( N1 Addr1 - )
     \ write 16 Bit
```

Sie setzen voraus, daß das FLASH auf FF gelöscht ist. Es muß aber zum Schreiben immer noch das entsprechende Teilprogramm nach FCODE ins RAM kopiert werden.

Einzelne Seiten löscht:

```
ERASE \ ( Addr1 - )
```

Addr1 muß eine gültige FLASH-Adresse innerhalb der Seite sein. Bei den Seiten für FLBPR und die Vektoren sind nämlich einige Adressen

## Tabelle 2: Versionsmeldung

```
nanoFORTH-08 Vd.d (c) Rafael Deliano 20dd SN= hhhh  
hhhh hhhh hhhh hhhh hhhh  
App SYS APP SYS Vektor  
EE EE FLASH FLASH
```

nicht mit FLASH belegt.

Den gesamten Programmspeicher der Applikation löscht:

```
0 FORGET
```

Um das spätere Löschen von FLASH zu vereinfachen richtet der Marker ^ den Programmanfang auf das nächste 128 Byte Speichersegment aus.

Um den ungenutzten Speicher wiederzugewinnen ist es deshalb sinnvoll nach Ende der Tests ohne Marker nochmal zu compilieren. Mehr zu Markern ab Seite 2.8.

## Schreibschutz mit Jumper

```
JUMPER? \ ( - Flag )  
          \ Flag = 1 : steckt
```

Um irrtümliche Veränderung von nanoFORTH zu verhindern, ist die Software so ausgelegt, daß Schreiben von FLASH oberhalb B800 nur bei gestecktem Jumper ( vgl. Bild 1 ) ausgeführt wird. Damit das zum Einstellen der Vektoren ab FFDC nicht nötig ist, zeigen diese ins ungeschützte EEPROM SYS-EE und können dort über einen Sprungbefehl leicht auf Applikationsprogramme umgeleitet werden.

## Re-Initialisierung

Oft wird man das System so konfigurieren, daß es nach Reset direkt eine Applikation startet. Um aus diesem Zustand wieder herauszukommen ignoriert das System bei Reset mit gestecktem Jumper jede Applikation und geht in Terminalmodus. Takt wird dabei auf 2,45 MHz eingestellt, sodaß auch Betrieb mit 3V Versorgung möglich ist.

Um den Auslieferungszustand wiederherzustellen kann man durch den Befehl SYSEE! das SYS-EEPROM mit den ursprünglichen

Einstellungen laden. Funktioniert nur bei gestecktem Jumper.

Mit 0 FORGET kann man den Applikationsspeicher löschen. Und mit APP-EE ERASE das Applikations-EEPROM.

## Prüfsumme

```
VERSION \ ( - )
```

druckt eine Textzeile ( Tabelle 2 ) und 6 Hexzahlen aus.

SN ist die Seriennummer. Sie ist für den Kunden individuell, aber für nachgekaupte Controller gleichen Typs identisch.

Die folgenden Werte sind die Prüfsummen über Speicherbereiche:

```
APP-EEPROM  
SYS-EEPROM  
APP-FLASH  
nanoFORTH  
Vektoren
```

Zur Berechnung wird intern ein Prüfsummenbefehl verwendet der nach dem verwendeten Verfahren benannt ist:

```
FLETCHER  
\ ( Addr1 Addr2 - hhhh )  
\ Addr1: last byte  
\ Addr2: first byte
```

# SYS-EE

Während das Anwendungs-EEPROM im Auslieferungszustand leer auf FF eingestellt ist, enthält das System-EEPROM die in Tabelle 3 aufgeführten Konstanten.

## Takt

Im Auslieferungszustand wird 2,45 MHz Busfrequenz verwendet, weil der Controller damit auch bei 3V Versorgungsspannung läuft. Die dafür intern nötige Taktfrequenz von 9,8 MHz wird aus dem 32kHz Quarz über eine PLL erzeugt. Aus der Busfrequenz wird gleichzeitig die Baudrate von 9600 Baud für die UART abgeleitet. Die Busfrequenz beeinflusst auch den in nanoFORTH vorhandenen Befehl MSEC der auf einer Verzögerungsschleife beruht.

Man kann durch Änderung der Konstanten im EEPROM die Taktfrequenz verändern. Vgl. Seite 1.6.

Bei Reset mit gestecktem Jumper wird unabhängig vom EEPROM fest die Taktfrequenz 2,45MHz und 9600 Baud eingestellt.

## CONFIG

Dieses Register des Controllers besteht aus 2 Bytes von denen jedes nach Reset nur einmal geschrieben werden kann. Das System muß nach Reset beide Werte schreiben um den Takt der UART auf Bustakt einzustellen und den Watchdog abzuwürgen. Um für die nötige Flexibilität zu sorgen werden die beiden Bytes aus dem EEPROM gelesen.

## Watchdog

Normalerweise deaktiviert. Wenn er in einer Applikation benötigt wird muß er in CONFIG1 freigeschaltet werden. Dabei kann auch die Länge des Timeouts gewählt werden:

```
1F B% 00001000 #. MOV,  
\ kurzer Timeout
```

Tabelle 3: Voreinstellungen  
SYS-EE

8080	00	PCTL'	Clock
8081	80	PBWC'	
8082	01	PMSH'	
8083	2C	PMSL'	
8084	80	PMRS'	
8085	01	PMDS'	
8086	0132	MSEC'	MSEC
8088	20	BAUD'	UART
8089	40	(KEY?)'	Parser
808A	01	CONFIG1'	CONFIG
808B	01	CONFIG2'	
808C	hhhh	LATEST'	FORTH
808E	hhhh	>ZV'	
8090	0140	>V'	
8092	00	LF'	
80C0	80 80 80	timebase	
80C3	80 80 80	ADC	
80C6	80 80 80	keyboard	
80C9	80 80 80	SCI transmit	
80CC	80 80 80	SCI receive	
80CF	80 80 80	SCI error	
80D2	80 80 80	SPI transmit	
80D5	80 80 80	SPI receive	
80D8	80 80 80	TIM2 overflow	
80DB	80 80 80	TIM2 chan. 1	
80DE	80 80 80	TIM2 chan. 0	
80E1	80 80 80	TIM1 overflow	
80E4	80 80 80	TIM1 chan. 1	
80E7	80 80 80	TIM1 chan. 0	
80EA	80 80 80	PLL	
80ED	80 80 80	IRQ	
80F0	80 80 80	SWI	
80F3	FF FF FF	Autostart	

```
1F B% 10001001 #. MOV,
\ langer Timeout
```

Kurzer Timeout entspricht ca. 2<sup>13</sup> Takte der CGMXCLK, d.h. 250msec mit 32kHz Quarz. Langer Timeout 2<sup>18</sup> fast 8,2sec.

Rücksetzen des Watchdogs erfolgt durch rechtzeitiges Schreiben von Adresse FFFF mit beliebigem Inhalt. In FORTH vorzugsweise inline so compilieren:

```
[CODE H% FFFF STA, CODE]
```

## FORTH

Aus der EEPROM-Konstante LATEST' wird nach Reset die Variable LATEST ins RAM geladen. Diese enthält die oberste NFA-Adresse der Befehlsliste. >ZV und >V sind die Zeiger auf das nächste freie Byte für ZVARIABLE bzw. VARIABLE. Auch sie werden nach Reset aus dem EEPROM initialisiert. Durch die Definition von Befehlen und Variablen ändert man diese Zeiger im RAM. Man kann die Werte durch den Befehl

```
SAVE \ ( -- )
```

vom RAM ins EEPROM zu sichern, wenn der Test der Software erfolgreich abgeschlossen ist. Auch der Befehl | sichert die Zeiger, jedoch am Ende des Applikationsprogramms.

## Interruptvektoren

Um die Vektoren problemlos auf Interruptprogramme verbiegen zu können zeigen sie jeweils auf ein 3 Byte Feld im EEPROM. Dieses ist ursprünglich mit RTI, -Opcodes 80 gefüllt. Zur Aktivierung ersetzt man das erste Byte mit CC und schreibt an das zweite Byte die absolute Adresse des Ziels. Z.B. für 80C0 so:

```
\ ( addr - )
CC 80C0 EC! 80C1 E!
```

Man hat damit diesen Befehl erzeugt:

```
hhhh JMP,
```

## Autostart

Beim letzten Vektor Autostart wird nach Reset überprüft ob 80F3 gelöscht FF ist. Wenn nicht wird diese Adresse mit

```
80F3 JSR,
```

als Unterprogramm angesprungen. Man kann von hier mit einem JMP, direkt eine Applikation aufrufen und niemehr zu FORTH zurückkehren.

# Reset

Es gibt zwei Resetbefehle: COLD und WARM. Letzterer führt nur eine Teilinitialisierung durch und ist der Einsprung des Befehls ERROR. Er wird hier nicht näher behandelt.

Bild 5 zeigt den Ablauf bei COLD wie er nach Reset ausgeführt wird.

Zuerst wird der Stackpointer der CPU initialisiert. Dann wird geprüft ob der Jumper steckt und ob im System-EEPROM ein Sprungbefehl am Autostart-Vektor vorhanden ist. In diesem Fall wird die Adresse mit JSR, angesprungen.

Nach dem Einschalten des LEDs wird geprüft ob das System-EEPROM schon geladen ist. Wenn nicht, werden Konstanten für das CONFIG-Register und die Takteinstellung 2,45 MHz in die Register des Controllers geladen. Ansonsten wird nur das CONFIG-Register aus dem SYS-EEPROM geladen.

Danach wird das RAM auf 00 gelöscht und dann Konstanten für FORTH geladen. Wenn das System-EEPROM nicht geladen ist, wird es als nächstes mit festen Werten komplett initialisiert. Wenn der Jumper nicht steckt wird die Takteinstellung aus dem EEPROM geladen.

Nun werden weitere Konstanten für FORTH aus dem System-EEPROM übernommen. Nach Blinken des LEDs wird die UART initialisiert und die Resetmeldung aufs Terminal ausgegeben.

Darauffolgend wird der Befehl WARM abgearbeitet, der die Stackpointer und einige Systemvariablen für Compiler und Assembler nochmal initialisiert. Denn beim Auftreten einer Fehlermeldung ERROR will man nicht, daß ein kompletter Reset ausgeführt wird der das RAM zerstört.

Sind COLD und WARM durchlaufen, befindet sich das System in der Endlosschleife des Interpreters.

Wenn der Jumper nicht gesteckt ist kann über den Autostart-Vektor im SYS-EEPROM optional auch ein Programm aufgerufen werden das I/O passend für eine Applikation initiali-

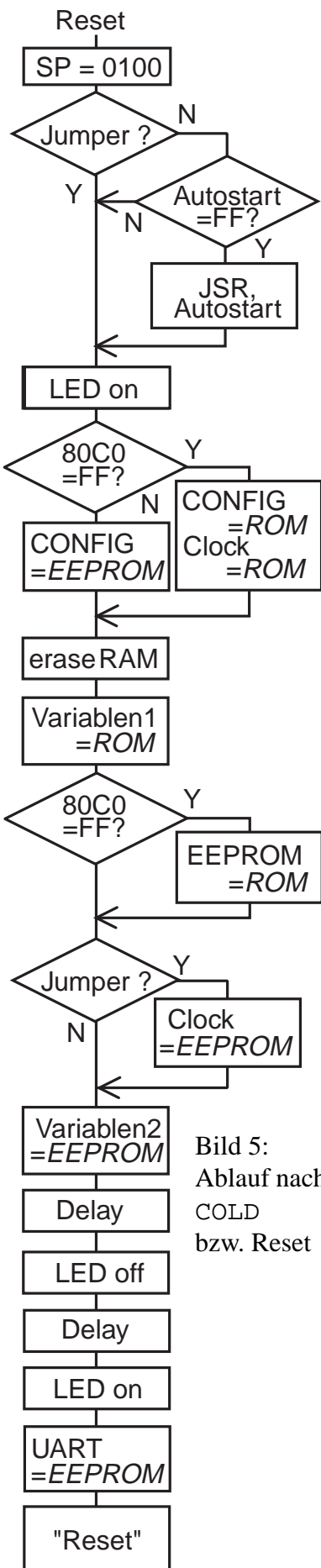


Bild 5:  
Ablauf nach  
COLD  
bzw. Reset

siert. Da der Aufruf über JSR, erfolgte ist über RTS, eine Rückkehr in den Interpreter möglich. Das ist in der Testphase oft nützlich.

## Bustakt ändern

Außer der 2,45 MHz Busfrequenz sind durch die PLL zwar von DC - 8 MHz viele Einstellungen möglich. Da die UART die 9600 Baud normalerweise aus der Busfrequenz ableitet schränkt sich die Zahl der sinnvollen Werte aber deutlich ein (Tabelle 4).

Man kann diese Werte für Tests nicht direkt in die Register des Controllers schreiben. Es empfiehlt sich ein Programm das aber auch in FBUF im RAM laufen kann (Listing 1). Anders als die EEPROM-Konstante (KEY?) die bei Reset in die Variable UDLY kopiert wird, greift der Befehl

MSEC immer direkt auf die Konstante MSEC' im EEPROM zu. Er läuft dann also erstmal mit falscher Verzögerungszeit, wenn man das EEPROM nicht ändert.

Die Schreibroutinen in nanoFORTH für das FLASH sind auf 2,45 MHz Bustakt ausgelegt. Es muß also bei dieser Frequenz compiliert werden, danach kann man für Tests die Frequenz umschalten. Applikationen die das FLASH nicht ändern oder dafür eigene Routinen mit anderem Timing verwenden können auch direkt aus dem Reset auf anderen Frequenzen laufen.

Tabelle 4: Werte für EEPROM geänderten Bustakt

2,45	4,91	7,37	8,00	MHz	
00	02	02	02	PCTL'	E = 1
80	80	80	80	PBWC'	Auto
01	02	03	03	PMSH'	N = 012C
2C	58	84	D1	PMSL'	
80	80	C0	D0	PMRS'	L = 80
01	01	01	01	PMDS'	R = 1
306d	612d	918d	1000d	MSEC'	
20	21	12	30	BAUD'	
64d	128d	192d	209d	(KEY?) bzw.	UDLY

Listing 1: Bustakt umschalten: Testversion, EEPROM unverändert

```

<|
FBUF >C !
:CODE 8MHZ
        36 5 MBC, \ PLL-OFF = 0
00 #. LDA, 36 STA,
02 #. LDA, 36 STA,
D1 #. LDA, 39 STA,
03 #. LDA, 38 STA,
D0 #. LDA, 3A STA,
01 #. LDA, 3B STA,
        36 5 MBS, \ PLL-ON = 1 : PLL on
        37 7 MBS,
3 $: 3 $ 37 6 BBC, \ wait for PLL to lock in
        36 4 MBS,
D% 209 #. LDA, UDLY STA, \ (KEY?)
        30 #. LDA, 19 STA, \ BAUD-rate
RTS,
CODE; |>
  
```

# Programmiergerät

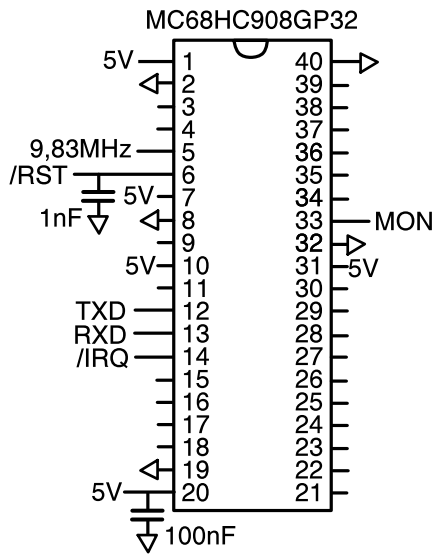
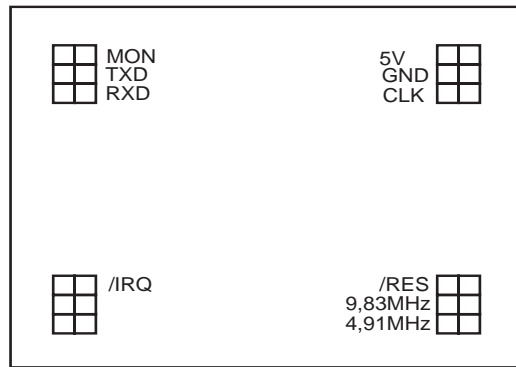


Bild 3: Stromlauf Adapter



Die von Motorola empfohlene Methode den Controller zu programmieren ist die Software von PME Microcomputer Systems die man kostenlos von deren Webseite ( [www.pemicro.com](http://www.pemicro.com) ) laden kann. Läuft auf Windows 95/98/NT. Sowie eine passende Schaltung die über V24 vom PC mit 9600 Baud aus angesteuert wird ( Bild 4 ).

## Hardware

Da es den Controller auch in anderen Gehäusen als DIL40 gibt, und man die Grundschaltung auch für andere Derivate verwenden kann, empfiehlt es sich die Schaltung in Adapter ( Bild 3 ) und Grundschaltung ( Bild 1 ) zu teilen. Eine mögliche Standardpinbelegung der Schnittstelle zeigt ( Bild 2 ), wobei einige Signale hier nicht vorhanden

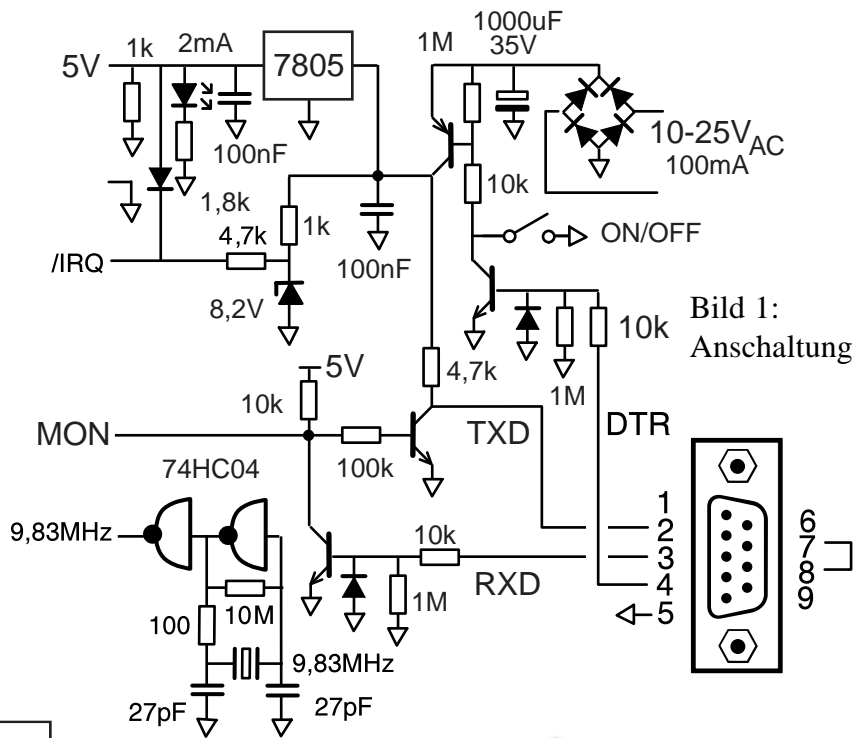


Bild 1: Anschaltung

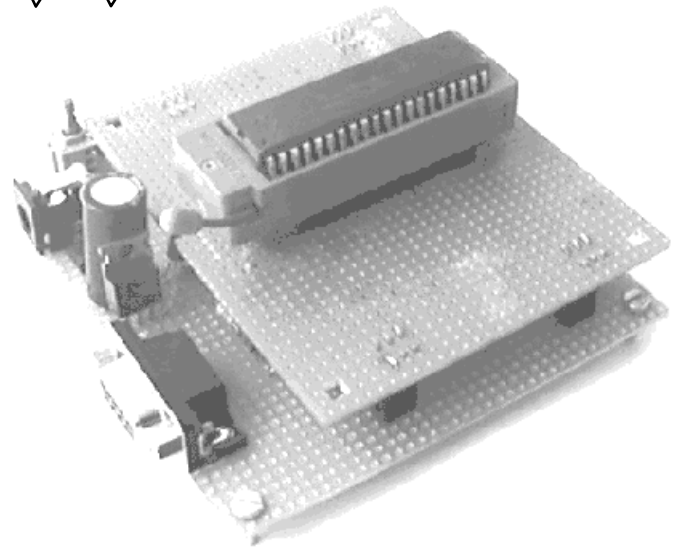


Bild 2: Pinbelegung Adapter/Anschaltung

sind. In der Schaltung kann der DTR-Pin die Versorgungsspannung des Controllers steuern ( „Class I“ in PME-Terminologie ) was zum gezielten Auslösen von Resets recht praktisch ist. Als Stromversorgung eignet sich ein kleines Steckernetzteil das natürlich auch DC-Spannung liefern kann. Der Anschluß ist polungssicher.

## Software

Wenn der Controller bereits programmiert ist kommt man nur ein wenn man den Security Code angibt. Für nanoFORTH 1.0 also:  
 80 EA 80 ED 80 F0 80 F3

Für MassErase, also zum Löschen kommt man auch ohne rein.

Als nächstes muß man mit „choose module“ den Code aus 908\_gp32.08p in den Controller laden. Erst danach hat man Zugriff auf die üblichen Funktionen eines Programmiergeräts.

Ersteinmal wird man mit „Upload Module“ eine Sicherheitskopie von nanoFORTH auslesen. Danach kann man dann ungeschützte Versionen von nanoFORTH duplizieren. Bzw. auf demolierte Controller nach MassErase nanoFORTH neu aufspielen. Für geschützte Versionen ist jedoch ein anderer Security Code nötig.

# Sicherung

Nach Abschluß der Entwicklung sind verschiedene Maßnahmen zum Schutz der Software erforderlich. Diese werden durch eine Veränderung des FLASH oberhalb nanoFORTH bewirkt, es muß also jeweils der Jumper gesteckt sein.

Fehlgeschlagene Änderungen können oft nur noch durch MassErase behoben werden, man sollte sich also nur daran versuchen wenn man Programmieradapter und eine Sicherheitskopie von nanoFORTH hat.

## FLBPR

Das Schreiben dieses Bytes im FLASH des Controllers mit

```
FLBPR F! \ ( UC1 - )
```

sorgt für irreversiblen Schreibschutz. Dagegen hilft nur noch MassErase auf Programmiergerät der das komplette FLASH inklusive nanoFORTH löscht. Die Startadresse des geschützten Bereichs ist in 128 Byte

Tabelle 5: Übliche Einstellungen für FLBPR

00	FLBPR!	\ komplettes FLASH geschützt
01	FLBPR!	\ APP-EE ungeschützt
02	FLBPR!	\ APP-EE, SYS-EE ungeschützt
70	FLBPR!	\ nur FORTH, Vektoren geschützt

Seiten einstellbar, gängige Werte zeigt Tabelle 5.

## Ausleseschutz

Wenn Geräte nicht für Eigenbedarf bestimmt sind muß man nanoFORTH und wohl auch die Applikation vor dem Zugriff unberechtigter Dritter schützen. Es gibt zwei Zugangswege.

Erstens über Motorolas Monitormodus. Dieser verwendet die Vektorbytes FFF6 - FFFD als Zugangscodes. Zweitens konventionell über V24 im Interpretermodus von nanoFORTH.

In ersterem Fall muß man also Interruptvektoren die normalerweise aufs SYS-EEPROM zeigen verändern. In letzterem Fall wurde die Sperre deshalb analog ausgelegt. In COLD wird geprüft ob die Adresse FFFC den Wert 80F3 enthält. Wenn das nicht mehr der Fall ist, stoppt COLD und druckt kontinuierlich RESET.

Typisch wird man die betroffenen Vektoren so verändern, daß sie nicht den Umweg über SYS-EEPROM machen sondern direkt auf Interruptprogramme springen. Bzw. wenn sich nicht benutzt sind, auf einen Dummy-Interrupt der nur aus dem RTI, - Opcode besteht.

Um diese Änderungen einfach durchzuführen liest der Befehl

```
VECT@ ( - )
```

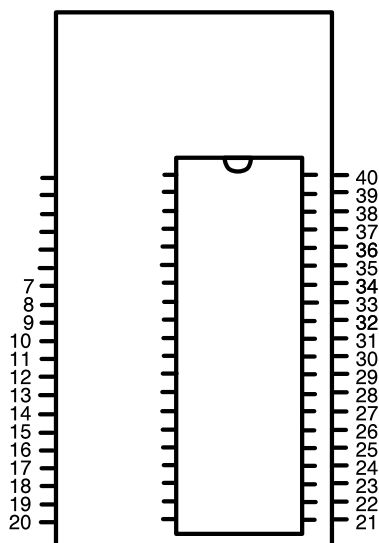
die Vektoren FFC0 - FFFF nach FBUF an 01C0 - 023F. Dort kann man dann den Patch manuell beliebig vornehmen. Dann sollte man mit 01C0 DUMP die neuen Einstellungen auf Diskette mitschreiben, denn sie sind als Zugangscodes für den Monitormodus nützlich. Der Befehl

```
VECT! ( - )
```

führt das Rückschreiben auf die Vektoren durch.

# Breadboards

In vielen Fällen ist für Fädeldrahtaufbauten eine Huckepackplatine nützlich die die Minimalbeschaltung auf Bild 1 enthält.



1.8

Ihre beiden einreihigen Buchsenleisten entsprechen in der Pinbelegung dem Controller, aber Pins 1-6 sind nicht belegt. Auf der Grundplatte befindet sich die Hardware der Applikation und die Stromversorgung.

Damit spart man sich bei folgenden Applikationen das Fädeln der Controller-Beschaltung, da man dann das Modul komplett übernehmen kann.

